The AVR iBoard Guide

Package Contents:

Opened the package?? Here's a checklist of the things you must have received.

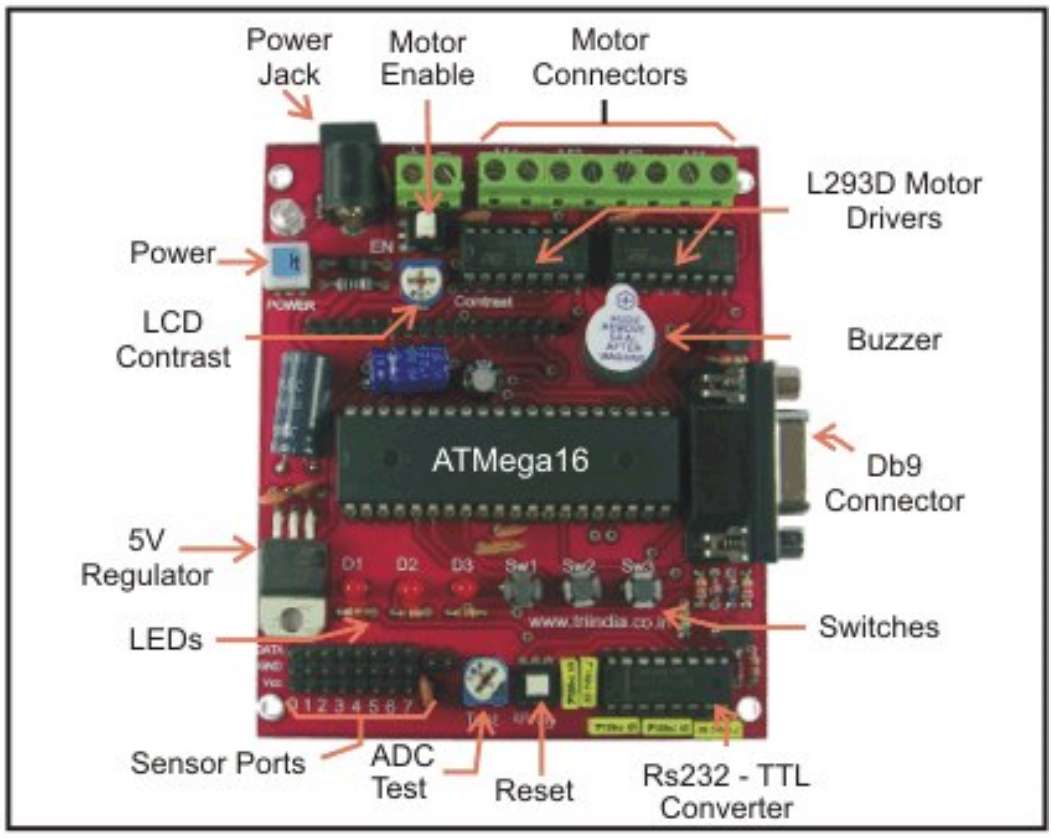
1. Controller board
2. ATMega16 micro controller
3. 16x2 LCD
4. Serial Cable
5. Getting started CD

So are we ready to get started?? Then read on…

The AVR iBoard:

- Includes ATMEL ATMega 16 (Also supports ATMega32) Microcontroller with 16kB flash memory working at 16Mips
- In system programmable
- On board programmer
- On board regulated power supply
- On board motor drivers with current capacity of upto 600mA per channel
- Power indicator LED
- 2 on board Dual full H bridge motor driver for 2 stepper or 4 DC motors
- Separate ON/OFF switch for power and motor drivers
- 3 LEDs for status or debugging purposes
- 3 ON/OFF switches for external inputs/interrupts
- On board LCD connector(multipurpose port,which can be used for other applications also)

Maximum Input voltage - 16V
Minimum Input voltage - 7V



Parts identification:

Power On Switch: It's a basic push to on - push to off type switch.

IC 7805: It's a three terminal linear 5 volt regulator used to supply the microcontroller and other peripherals.
Motor Enable switch: This switch is used to enable/disable the motor driver chips hence in turn enabling/disabling the motors.
Reset Switch: This switch is used to reset the microcontroller.
MAX 232: This chip takes care of the voltage conversions needed to communicate with the PC's RS232 (Serial/ COM) port.
L293D: It is a 4 channel motor driver with 600mA of current per channel and has inbuilt clamp diodes. The board contains two such chips.
Potentiometer (Pot): The potentiometer is used to vary the contrast of the LCD.
Sensor port: At a time, 8 individual sensor modules can be connected to this port. The port also provides a 5V supply needed drive the sensors.
DB 9 connector: This is a 9 pin connector used to connect to the PC's COM port during programming or for general UART communications.
Switch array: Four general purpose switches are connected in the active-low configuration.
Crystal: A crystal sets the microcontroller's clock frequency to 11.0592 MHz.
Beeper: Connected in the active low mode, the beeper can easily be used to get audible feedbacks from the controller.
ADC Test: Short the two pins of the jumper next to the potentiometer and the potentiometer gets connected to the PortA 7.

Board Connection Details:

Port A

0…7	Sensor Connector

Port B

0…3	Motor Driver
4	Switch 1 (Active Low)
5,7	Programmer
6	LED D2 (Active High)

Port C

0	LCD Control Pin
1	LED D1 (Active High)
2	LCD Control Pin
3	Buzzer (Active High)
4…7	LCD Data Pins

Port D

0	USART RXD
1	USART TXD + LED D3
2,3	Switch 2,3
4…7	Motor Driver

Programming the AVR

AVR microcontrollers are fun, powerful, feature rich and among those microcontrollers that can be got up and running quickly without much effort. Secondly a lot of resources are available on the net thanks to the ever widening AVR community. Free ware programmers, compilers, projects, name it and you get it.

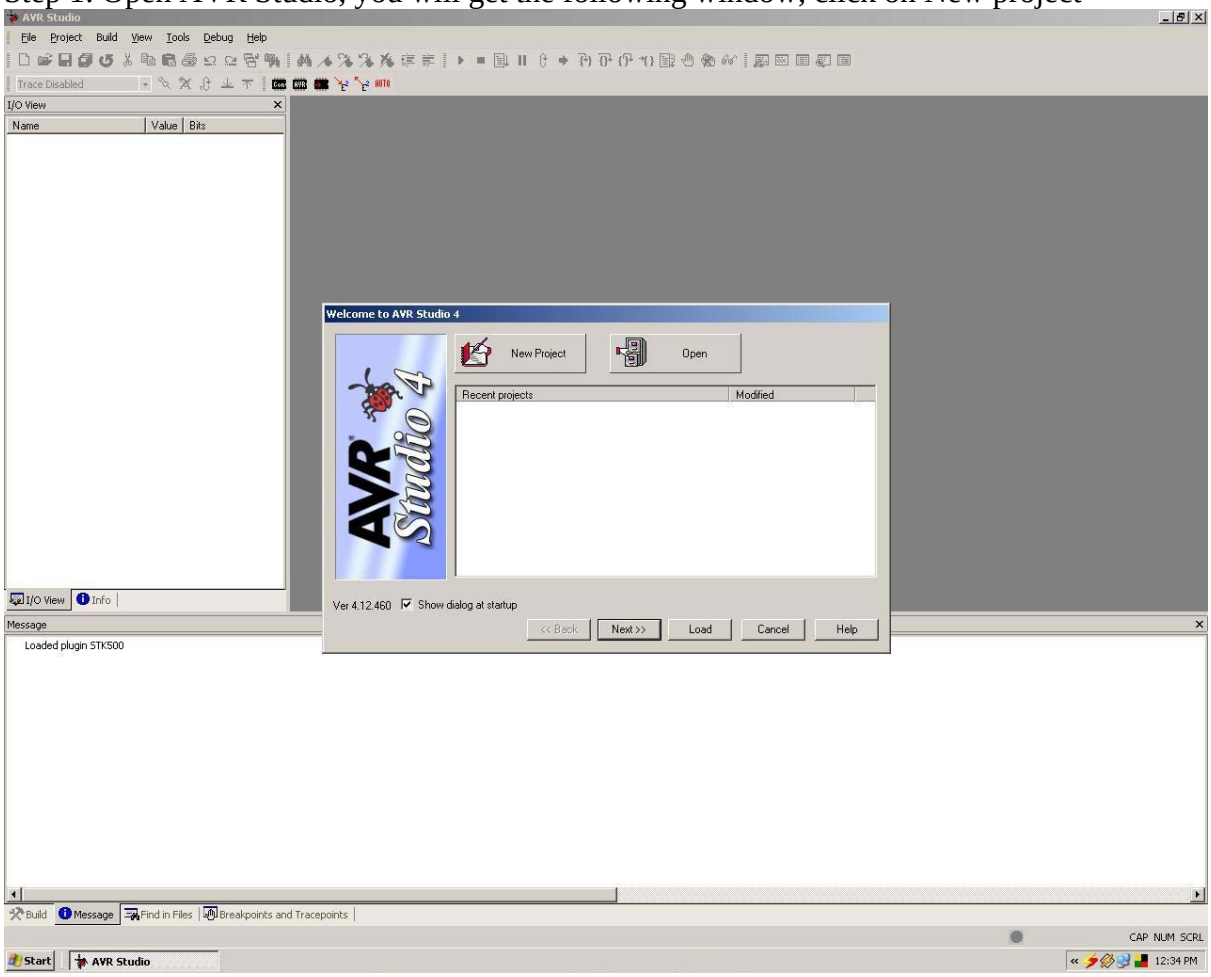
For our use we will be working with the following softwares
WinAVR - http://winavr.sourceforge.net/
AVR Studio - http://www.atmel.com
PonyProg - http://www.lancos.com/prog.html

Download and install the following softwares before we start of on our quest to use to the AVR.

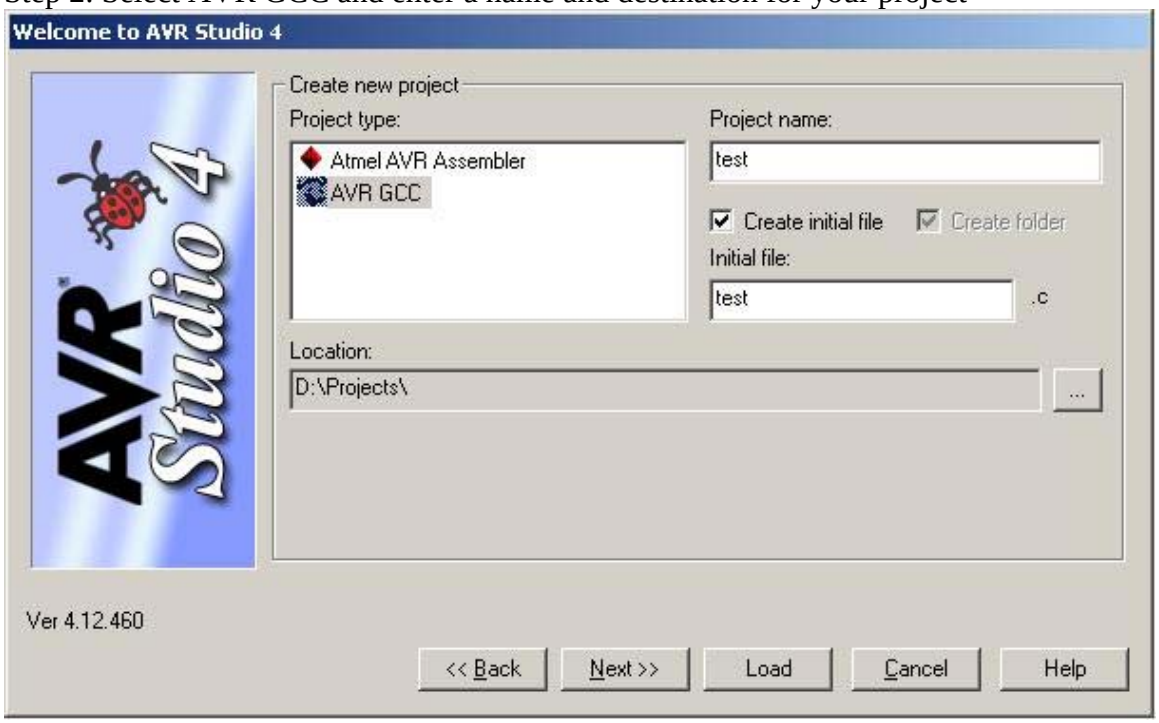
Checklist:

1. AVR board
2. Power supply
3. Serial Cable
4. Computer with the above softwares installed
5. Some patience :)

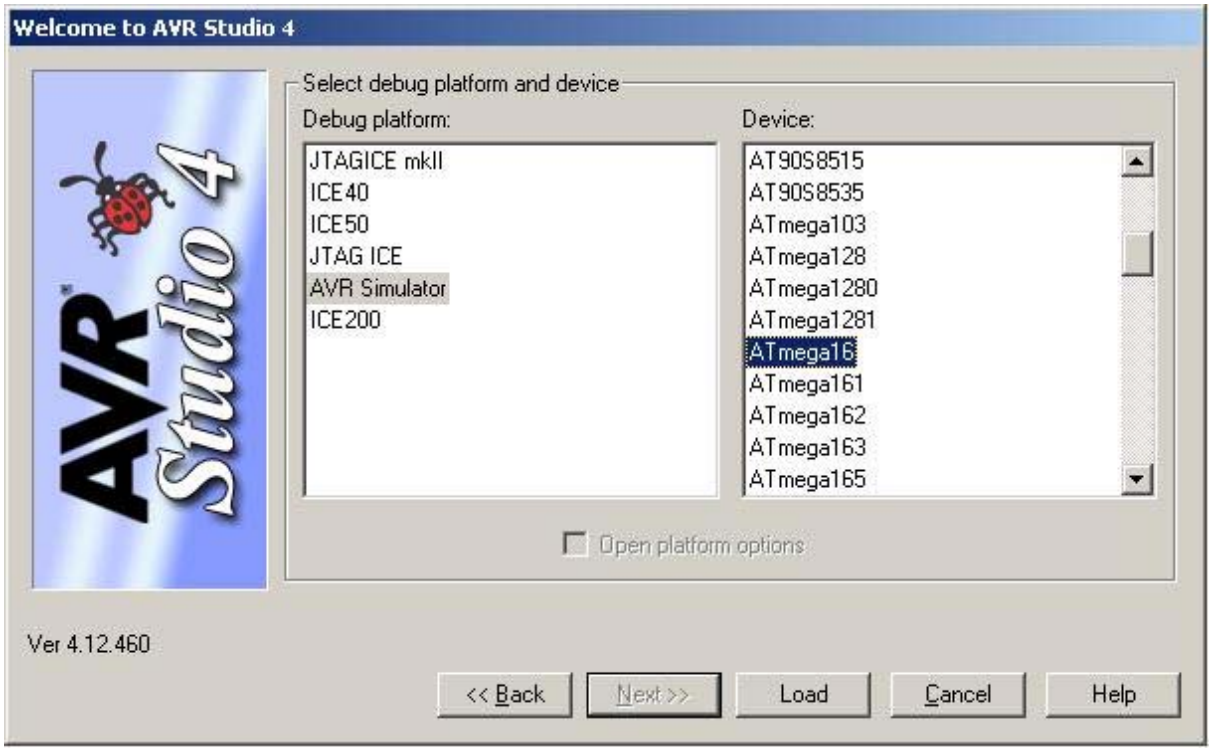
Step 1: Open AVR Studio, you will get the following window, click on New project



Step 2: Select AVR GCC and enter a name and destination for your project



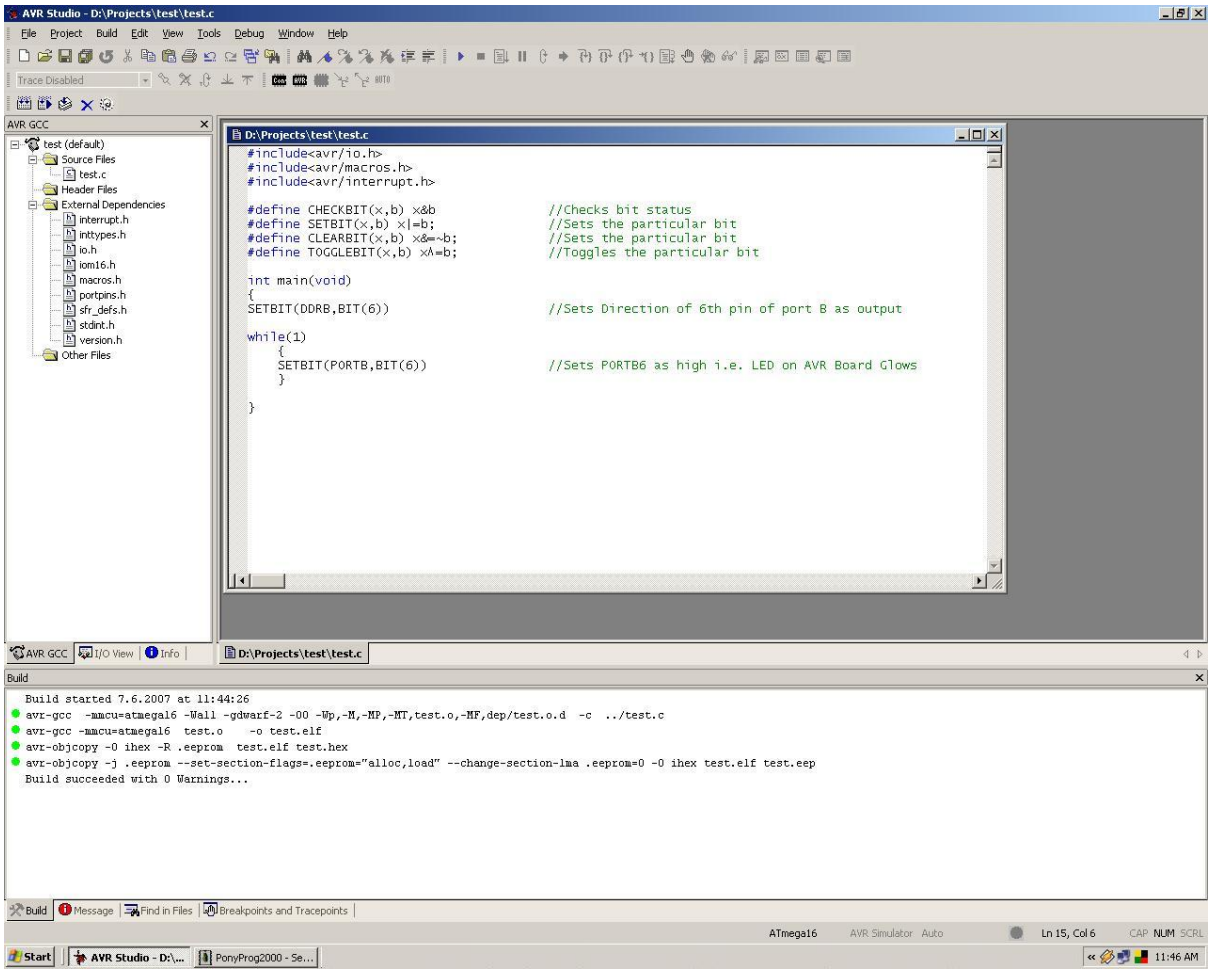
Step3: Select AVR simulator and the device to be used.



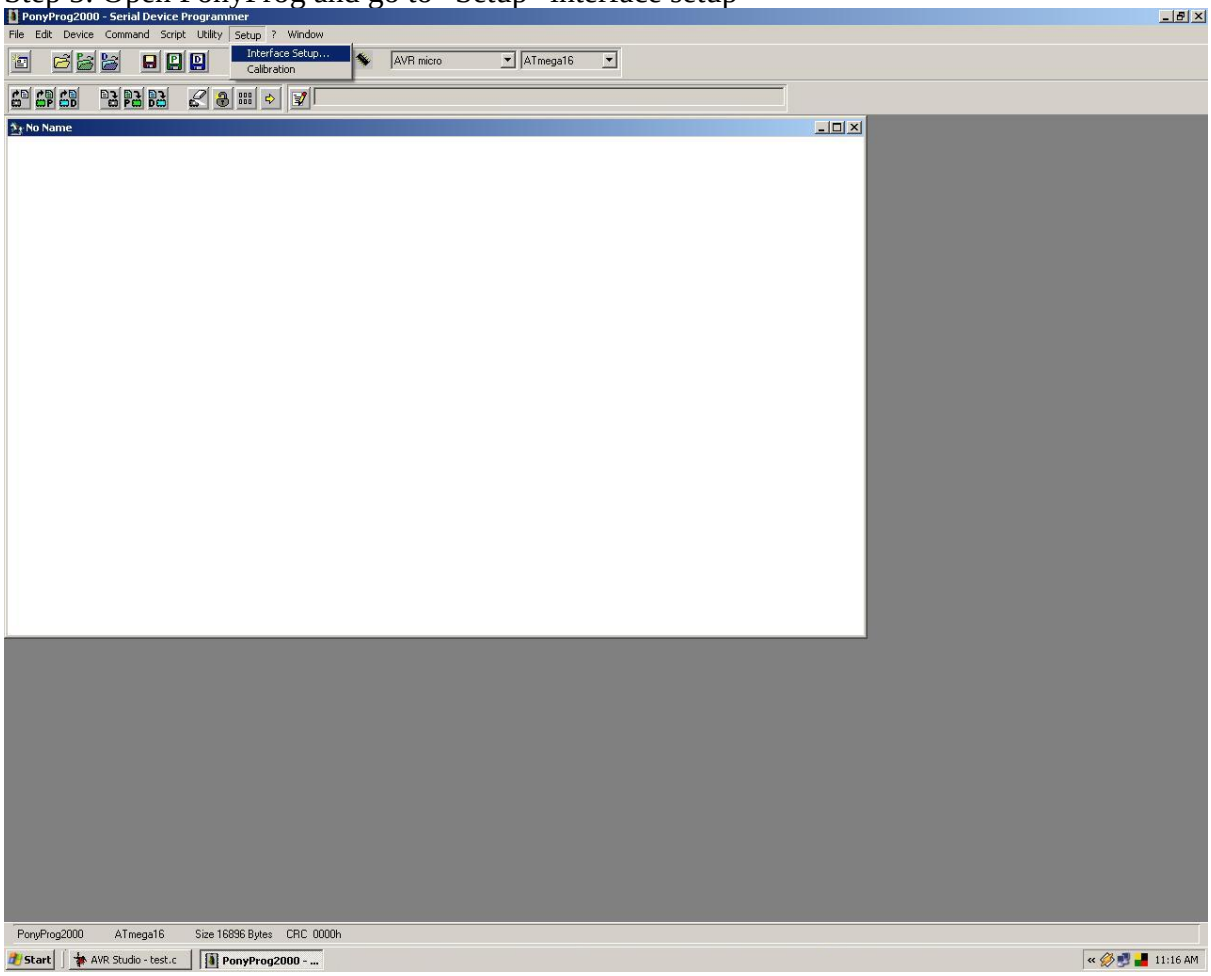
Step 4: Start writing your first program.

```
#include<avr/io.h>
#include<avr/interrupt.h>

/*Macros definition*/
#define BIT(x)          (1 << (x))
#define CHECKBIT(x,b) x&b                   //Checks bit status
#define SETBIT(x,b) x|=b;                   //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;                //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;                //Toggles the particular bit

int main(void)
{
SETBIT(DDRB,BIT(6))                         //Sets Direction of 6th pin of port B as output

while(1)
        {
        SETBIT(PORTB,BIT(6))      //Sets PORTB6 as high i.e. LED on AVR Board Glows
        }

}
```

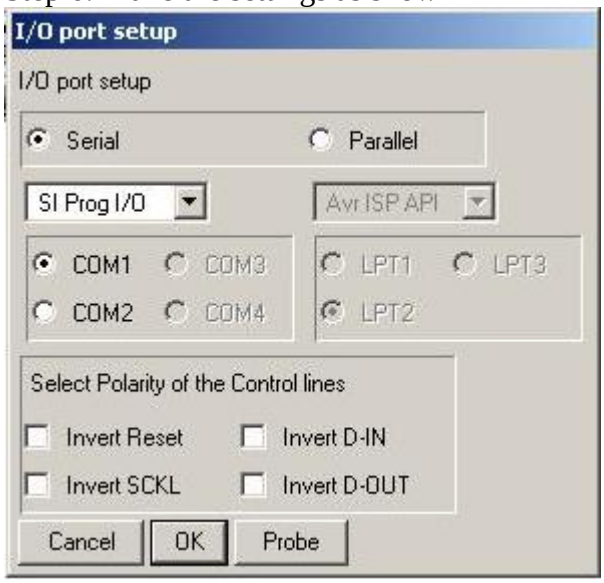
Goto “Build” menu and click on Build.



Step 5: Open PonyProg and go to “Setup” interface setup

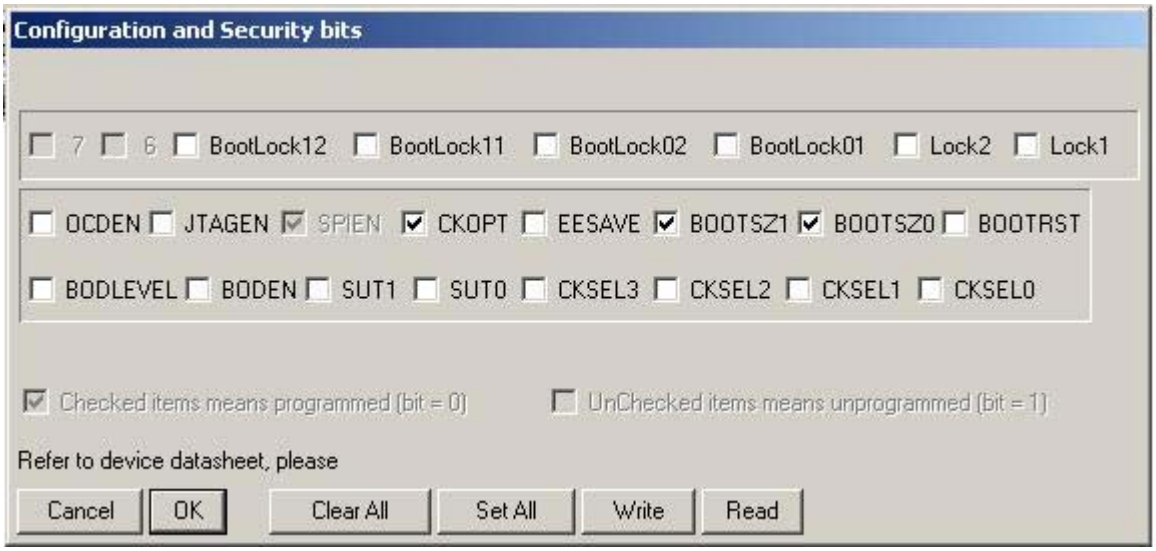


Step 6: Make the settings as shown

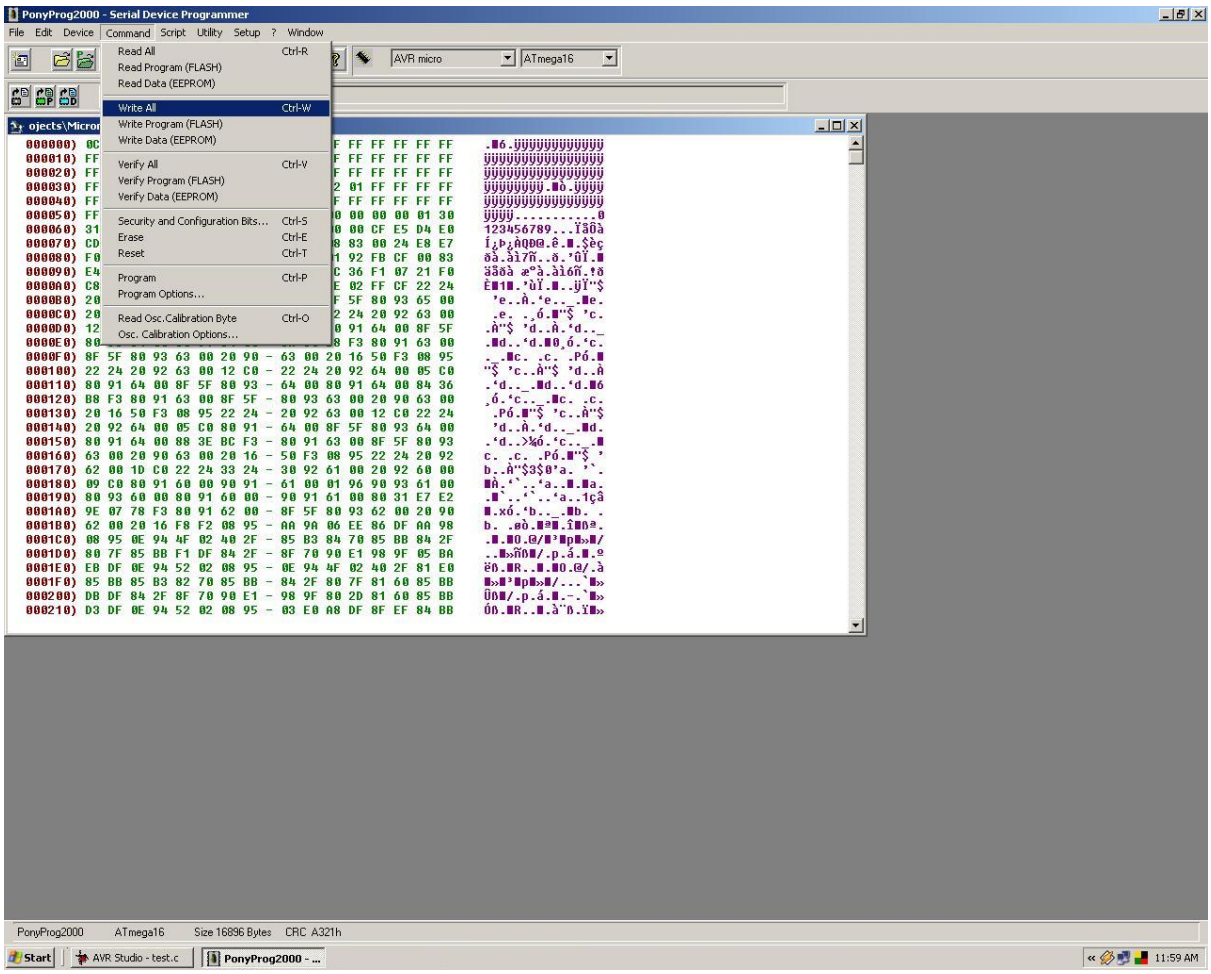


Step 7: Click on “Setup” menu and click on Calibration. A dialog box asking whether you want to run calibration now will appear. Click on “YES”

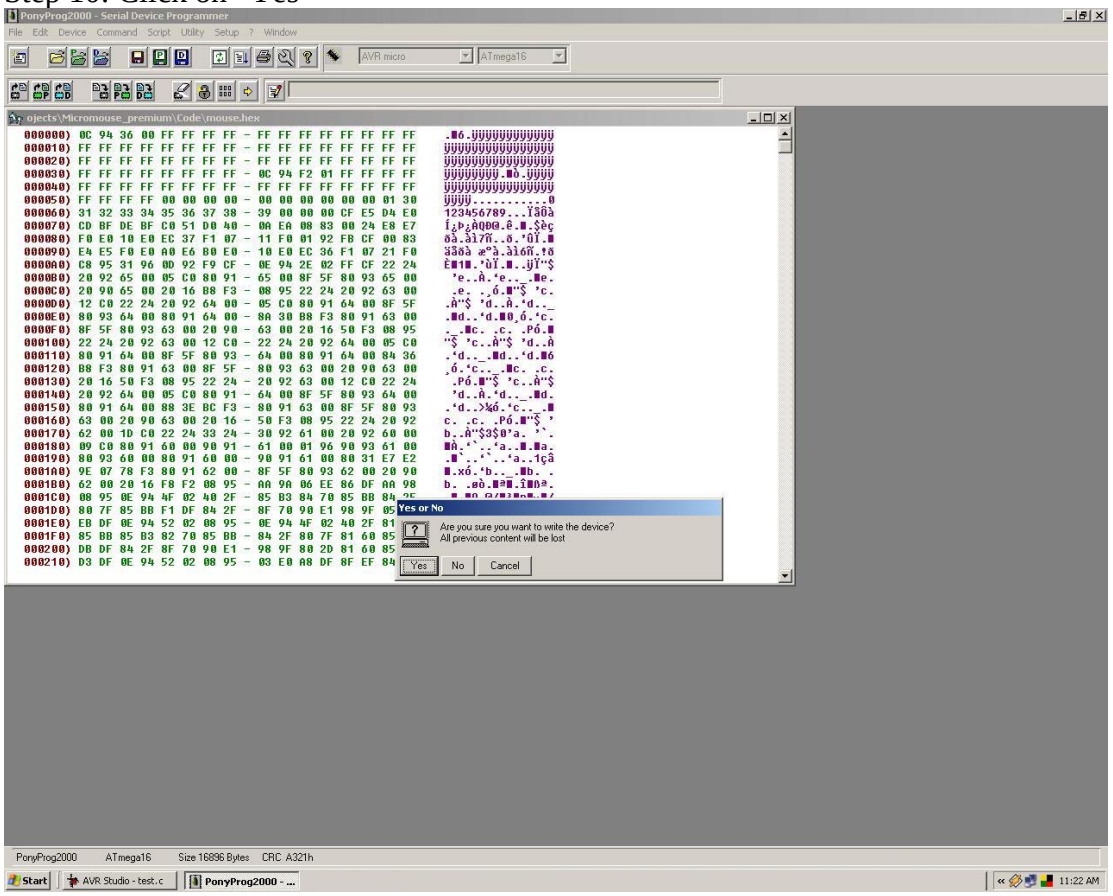
Step8: Click on “Security and Configuration Bits” .Click on read first. Then make the settings as shown. And click on write.

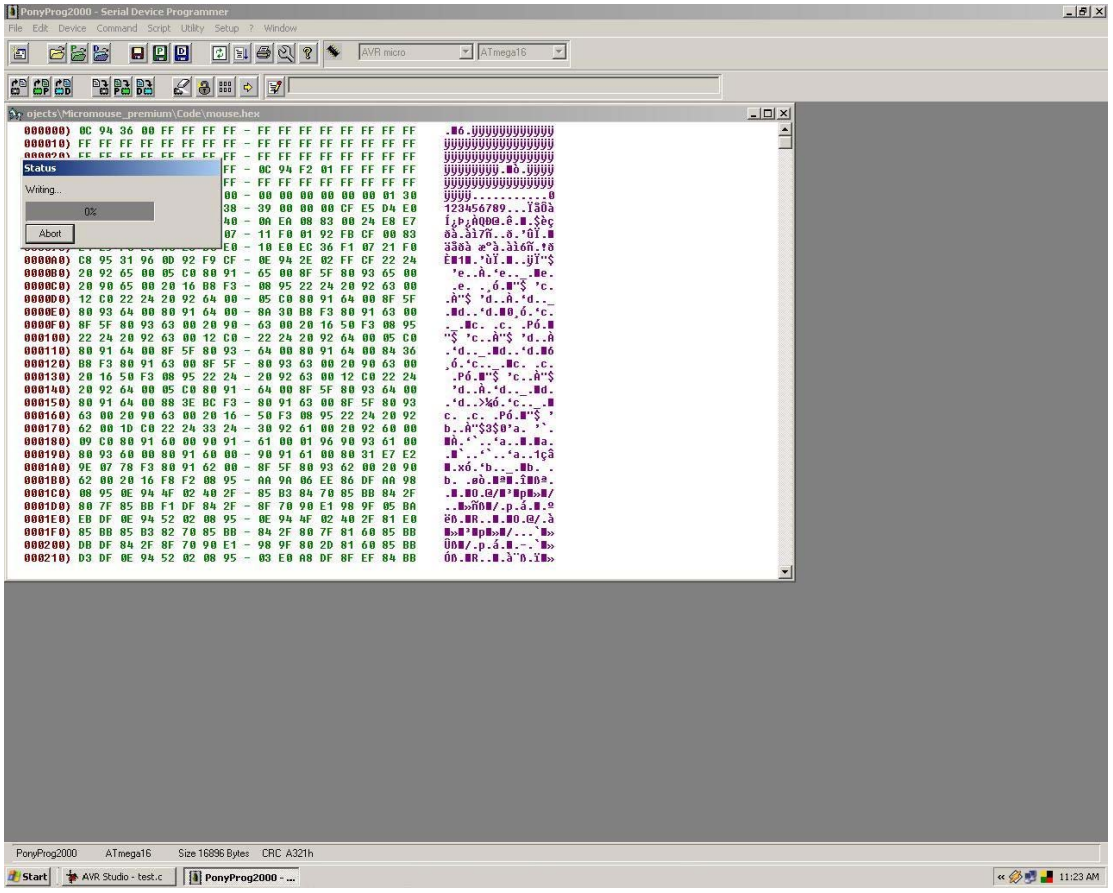


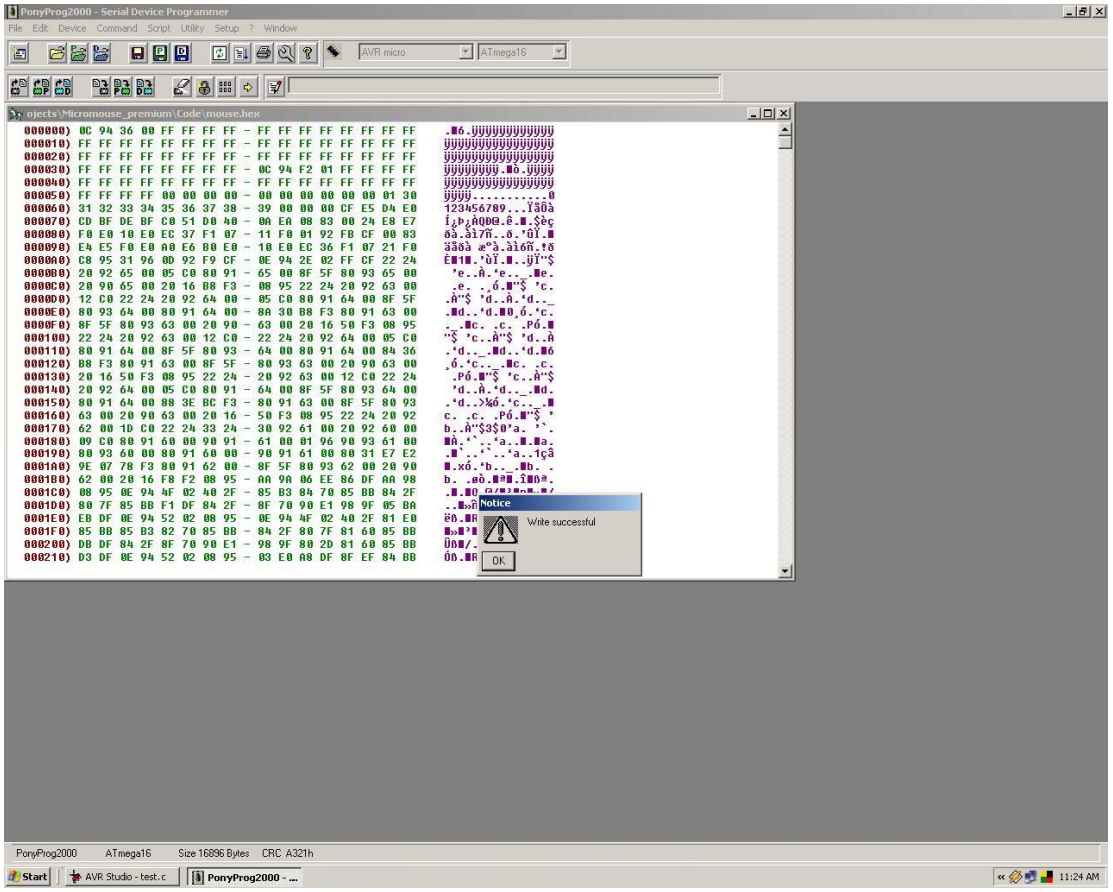
Step 9: Open the hex file that AVR studio generate. You will find it inside a directory called “Default” where the project was saved. Then go to command menu and click on write



Step 10: Click on “Yes”







Did you see the LED D2 light up?? If yes, Congratulations you have just climbed the first step.

If not, did you get the following error??

Check the following:
1. Is the power on?
2. Is the ‘Prog’ switch on the board pressed?
3. Is the serial cable connected?

If the answers to all this is ‘Yes’ check whether you have selected the correct device and the fusebit settings.
Happy Coding!!!

1. Program to Blink and LED

This code blinks the LED D2 on PORTB PIN6 i.e PB6 of the AVR board. The LED D2 is connected to PB6 on the microcontroller, and is wired so that it will be ON when the output is high and OFF when the output is low. The "special function registers" used in this program are:

DDRB -- Port B Data Direction Register

A high (1) value indicates output, low (0) is input.We set the 6th bit in DDRB to configure the PB6 pin as output pin.

PORTB -- Port B Data Register

This is used to set the output value for each of the eight port B pins.

WaitMs() is used to produce a delay.

```
#include<avr/io.h>
#include<avr/interrupt.h>

/*Macros definition*/
#define BIT(x)        (1 << (x))                    //Set a particular bit mask
#define CHECKBIT(x,b) x&b                  //Checks bit status
#define SETBIT(x,b) x|=b;                  //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;               //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;               //Toggles the particular bit

void WaitMs(unsigned int ms);

int main(void)
{
        SETBIT(DDRB,BIT(6))         //Sets Direction of 6th pin of port B as output
        while(1)
        {
                SETBIT(PORTB,BIT(6))        //Sets PORTB6 as high i.e. turn on led
                WaitMs(200);                //delay for 200ms(0.2s)
                CLEARBIT(PORTB,BIT(6))      //Clears PORTB6(PB6 pin goes low) i.e
                                            //turn off led
                WaitMs(200);                //delay for 200ms(0.2s)
        }
        return 0;
}

/* waits (pauses) for ms milliseconds (assumes clock at 16MHz) */
void WaitMs(unsigned int ms)
{
        int i;

        while (ms-- > 0)
        {
                /* 16380 (16k) clock cycles for 1ms; each time through loop
                   is 5 cycles (for loop control + nop) */
                for (i = 0; i < 3276; ++i)
                        asm("nop");
        }
}
```

2. **Program Using Switches and LEDs**

This code glows the LED D2 on PORTB PIN6 i.e PB6 of the AVR board, if switch1is turned on else it is off. The LED D2 is connected to PB6 on the microcontroller, and is wired so that it will be ON when the output is high and OFF when the output is low. The Switch S1 is connected to PB4. The "special function registers" used in this program are:

DDRB -- Port B Data Direction Register

A high (1) value indicates output, low (0) is input. We set the 6th bit in DDRB to configure the PB6 pin as output pin. We clear the 4th bit in DDRB to configure the PB4 pin as input pin.

PORTB -- Port B Data Register

This is used to set the output value for each of the eight port B pins.

PINB --- PORTB input register, used to read inputs in PORTB pins

```
#include<avr/io.h>
#include<avr/interrupt.h>

/*Macros definition*/
#define BIT(x)        (1 << (x))                    //Set a particular bit mask
#define CHECKBIT(x,b) (x&b)                         //Checks bit status
#define SETBIT(x,b) x|=b;                   //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;                //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;                //Toggles the particular bit

int main(void)
{
        SETBIT(DDRB,BIT(6))                 //Sets the 6th pin in DDRB register to
                                            //make PB6 a output pin
        CLEARBIT(DDRB,BIT(4))               //Clears the 4th pin in DDRB register to
                                            //make PB4 a input pin
        while(1)
        {
                if(CHECKBIT(PINB,BIT(4)))           //Check 4th bit in PINB register
                        SETBIT(PORTB,BIT(6))        //Sets PORTB6 as high i.e. LED on
                                                    //AVR Board Glows
                else
                        CLEARBIT(PORTB,BIT(6))      //Clears PORTB6, PB6 goes low i.e
                                                    //turn led off
        }
        return 0;
}
```

3. **Program to Run Motors**

Connect motors to the motor connectors. We will move them in the either directions. The motors are connected to the PortD 4-7 and PortB 0-3

```
#include<avr/io.h>
#include<avr/interrupt.h>

/*Macros definition*/
#define BIT(x)        (1 << (x))                    //Set a particular bit mask
#define CHECKBIT(x,b) (x&b)                         //Checks bit status
#define SETBIT(x,b) x|=b;                   //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;                //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;                //Toggles the particular bit

void delay(unsigned int time)
{
unsigned int i,j;
for(i=0;i<1000;i++)
        {
        for(j=0;j<time;j++){}
        }
}

int main(void)
{
        DDRB=0x0F;          //Sets lower nibble as output
        DDRD=0xF0;          //Sets lower nibble as output

        while(1)
        {
        PORTB=0x05;         //Move motors in one direction
        PORTD=0x50;
        delay(1000);
        PORTB=0x0A;         //Move motors in one direction
        PORTD=0xA0;
        delay(1000);
        }
        return 0;
}
```

4. **Program to use LCD module**

This header files have implementation of some common finctions of the LCD.
Include this header file in your C code. Call the lcd_init() function before calling any other functions to initialize the lcd.

lcd_init() ----- Call the lcd_init() function before calling any other functions to initialize the lcd.

lcd_cmd() ------ This function is used to give any command instructions to the LCD. For e.g lcd_cmd(0x01) will give the clear command.

lcd_char() ----- This function will display a single character on the LCD display. For example lcd_char(0x61) will display A. Again lcd_char('b') will display b.

lcd_string()---- This function will display a string. An example of this would be like lcd_string("This is AVR")

lcd_showvalue()- This will show a 3-digit decimal value on the LCD. For example if we give lcd_showvalue(0xFF) then 255 will be displayed.

lcd_gotoxy1()--- Will set the cursor at a particular position on LINE1 of the LCD. So lcd_gotoxy1(3) will set the cursor at the 3+1=4th column in the 1st line of the LCD.

lcd_gotoxy2()--- Will set the cursor at a particular position on LINE1 of the LCD. So lcd_gotoxy2(0) will set the cursor at the 0+1=1st column in the 2nd line of the LCD.

lcd_exit() ----- You may call this function after you are over with your LCD. While calling the lcd functions, there would be some changes to the PORTC. This restores the original PORTC configuration before calling the lcd_init() function. This is not absolutely necessary. You may or may not use this function.

```
#include<avr/io.h>
#include"LCD.h"

#define BIT(x)        (1 << (x))
#define CHECKBIT(x,b) (x&b)              //Checks bit status
#define SETBIT(x,b) x|=b;          //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;             //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;             //Toggles the particular bit


int main(void)
{
        lcd_init();
        lcd_string("This is My First LCD program");
        while(1)
        {}
        return 0;
}
```

5. **Program to communicate with the PC through UART**

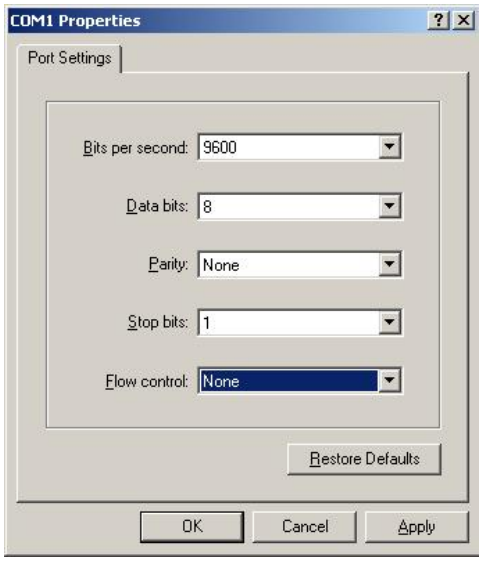
This code implements UART communication between your PC and AVR board. This is a simple ECHO program. So what you send from the Hyper terminal of the PC will be received by the micro controller and resend back, i.e it will echo back the data. In this program USART is configured to 9600bps, 8-bit character size and no parity. You need to configure the hyper terminal connection with these settings.

The "special function registers" used in this program are UBRR, UCSRA, UCSRB, UCSRC and UDR.

```
#include <avr/io.h>
#include<avr/interrupt.h>

/*Macros definition*/
#define BIT(x)        (1 << (x))          //Set a particular bit mask
#define CHECKBIT(x,b) x&b                 //Checks bit status
#define SETBIT(x,b) x|=b;                 //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;              //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;              //Toggles the particular bit

unsigned char Recv;

int main (void)
{
        //Initialize UART with 8-bit character sizes,no parity,1 stop bit
        UBRRL=0xA0;         //Write UBRRL register first
        UBRRH=0x01;         //then write the UBBRH register
        UCSRB=0x98;         //Set RXCIE, RXEN, TXEN bits
        UCSRC=0x86;         //Set URSEL,UCSZ0,UCSZ1

        sei();          //Enable Global Interrupt

        while(1)
        {
        }
}

//RX complete Interrup routine..put here all the code that you want to do with the
received data..
ISR(USART_RXC_vect)
{
        Recv=UDR;
        UDR=Recv;
}
```

Step 1: Enter a name for the connection

Step 2: Select the COM port and configure its properties



6. Program to use the ADC in ATMega

This code implements ADC conversion. The Test pot at PA7 (ADC7) is used to control the brightness of the led D2 (Make sure the jumper is inserted). The pot gives a variable voltage at the ADC7 pin. This is converted to a digital value. This digital value is then used to control the brightness of the LED. A simple PWM with delay routines is used. The "special function registers" for ADC are ADMUX, ADCSRA, ADCH, ADCL and SFIOR.

ADMUX is used to select which channel to use for ADC conversion. We used ADC7 here. It also has an ADLAR bit, by setting which the ADC data value becomes left adjusted.

ADCSRA contain bits to enable ADC,start ADC, select ADC prescale clock,etc.

ADCH and **ADC**L contains the ADC data..We used ADCH only coz we are satisfied with 8-bit resolution only.

SFIOR contains bits to configure different modes. Default value sets the ADC at free running mode.

The ADC ISR is executed when a ADC conversion completes. In the ISR, you can do whatever you like with the ADC converted value stored in ADCH:L At the end of the ISR routine you should set the ADSC (ADC start conversion) bit to again start the ADC conversion.

```
#include <avr/io.h>
#include<avr/interrupt.h>

/*Macros definition*/
#define BIT(x)      (1 << (x))          //Set a particular bit mask
#define CHECKBIT(x,b) x&b       //Checks bit status
#define SETBIT(x,b) x|=b;       //Sets the particular bit
#define CLEARBIT(x,b) x&=~b;          //Sets the particular bit
#define TOGGLEBIT(x,b) x^=b;          //Toggles the particular bit

unsigned char digital=100;

void Wait(unsigned char delay);

int main(void)
{
      //ADC Initialization
      ADMUX=0x27;       //We set ADLAR=1(left adjusted) and set the MUX bits
                        //to select ADC7 input.
      ADCSRA=0x8E;      //enable ADC and ADIE(Enable ADC interrupt)

      SETBIT(DDRB,BIT(6));      //Set the DDRB bit 6 to configure PB6 as output
      SETBIT(PORTB,BIT(6));     //Turn on the Led D2


      sei();                    //Enable Global Interrupt
      SETBIT(ADCSRA,BIT(6));    //Start ADC conversion

      while(1)
      {
            //Here we generate PWM using a simple delay routine,
            //ON time is proportional to the digital value recieved
            //from the ADC
```

```
                //ON TIME
                SETBIT(PORTB,BIT(6));       //turn on led
                Wait(digital);

                //OFF TIME
                CLEARBIT(PORTB,BIT(6));   //turn off led
                Wait(255-digital);

        }
        return 0;
}

//ADC conversion complete ISR
ISR(ADC_vect)
{
        digital=ADCH;           //Get the ADC converted value..Since we used Left
                                //adjusted, so we need to access the ADCH only

        SETBIT(ADCSRA,BIT(6));    //Start the next ADC conversion
}

//A delay routine
void Wait(unsigned char delay)
{
        unsigned char i;

        while (delay-- > 0)
        {
                for (i = 0; i < 65; ++i)
                        asm("nop");
        }
}
```